Automatic Monitoring of Software Requirements

Don Cohen, Martin S. Feather, K. Narayanaswamy & Stephen S. Fickas
Computing Services Support Solutions
5777 West Century Blvd., Suite 1230
Los Angeles, CA 90045-5600 USA
http://www.compsvcs.com
tel: +1 213 299 3136
donc@compsvcs.com, feather@compsvcs.com, swamy@compsvcs.com, fickas@cs.uoregon.edu

ABSTRACT

Automatic run-time monitoring of software systems' design- / purchase- / installation- time requirements and assumptions is a key step towards making those systems more robust, maintainable, and self-evolving.

A concise language has been designed to permit the convenient expression of a wide range of requirements and assumptions. A compiler automatically converts these expressions into run-time monitors to watch for, and report, all requirement and assumption violations.

The mechanism is applicable to systems which have not necessarily been designed with monitoring in mind, permits addition of further requirements and assumption monitoring on-the-fly, and emphasizes usability by a wide range of end-users.

Keywords
Monitoring, requirements, assumptions, expectations, maintenance, robustness, software evolution.

INTRODUCTION

It is common for the environment within which a software system resides to evolve. Assumptions of both the requirements that the system must fulfill, and the operating conditions in which the system will operate, may thus be rendered invalid over time. For example, the creator of a system may make assumptions about the way in which that system will typically be used, designing the system accordingly; the installer of a system may make assumptions about the computational resources that will be available, configuring the installation of the system accordingly. It is at run-time, however, that violations of these assumptions become manifest through the symptoms of frustrated users, squandered computational resources, and missed opportunities.

Software requirements monitoring extends the idea of resource monitoring (e.g., as practiced by administrators of computer networks) to the broader class of *requirements* and *assumptions* made by the designers, purchasers, installers and users of software systems.

The need for, and objectives of, requirements monitoring have been suggested previously both by ourselves [2] and others [3]. The latter group nicely identifies the approach as bridging gaps between different classes of people (e.g., designers and users) and across different times (e.g., design time and use time). Our approach has been to develop a system that, given expressions of requirements and assumptions, compiles those into run-time monitors to watch for, and generate notifications of, any violations. This system is the focus of the accompanying demonstration; its key characteristics are described here.

MONITORING DESIDERATA

Traditional programming languages and environments provide limited support for monitoring. Through constructs such as "assert" statements, "invariants", etc., they permit the programmer of the system to encode his/her assumptions. This capability is laborious to use (because the idioms of monitoring are not made conveniently expressible in general-purpose programming languages), and is restricted to monitoring queries that can be anticipated at coding time.

Monitoring would be practiced in a much broader range of circumstances if the following desiderata were met:

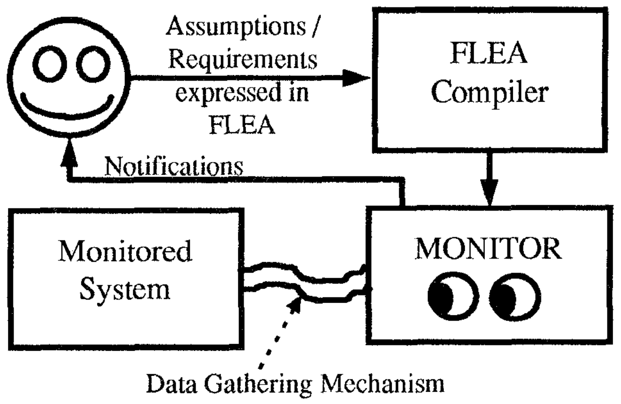
flexibility and convenience - a wide range of users' requirements and assumptions should be readily expressible

automatic compilation - these expressions of requirements and assumptions should be compiled automatically into reasonably efficient run-time monitoring code

applicable to "black-box" systems - monitoring should be applicable to systems which were not necessarily designed with monitoring in mind incremental - it should be possible to add monitoring queries "on-the-fly", as the monitored system continues to operate.

AMOS - ASSUMPTIONS MONITORING SYSTEM

Our approach to meeting these desiderata is embodied in a prototype monitoring system, AMOS. Its overall architecture is sketched below:



The user expresses his/her requirements and assumptions for monitoring in FLEA, a Formal Language for Expressing Assumptions. This small language provides a set of composable constructs tailored for the convenient expression of a wide range of monitoring concerns. Briefly, FLEA provides constructs for:

logical combination of events e.g., eventA **OR** eventB

sequences of events, e.g., eventA **THEN** eventB

counting and other simple statistical operations on events, e.g., **COUNT** [occurrences of] eventA

parameterized events, e.g., eventA(**x**)

time sensitive events, e.g., eventA **WITHIN** 60 seconds of eventB

threshold events, e.g., **START** count eventA > 10

Inspiration for FLEA's constructs was drawn from several sources, particularly GEM [5] and PPMS [4].

The FLEA compiler automatically converts FLEA expressions into run-time monitoring code. The underlying mechanism of the run-time monitor is that of AP5 [1], an active database (i.e., a database with "triggers" that examine each transaction to determine if and when their triggering conditions become true). The compiler takes advantage of the powerful capabilities of this underlying database, while shielding the end-user from its complexities.

The run-time monitoring code uses one of several generic data-gathering mechanisms to observe the interactions between the system being monitored. and that system's environment. For example, one data-gathering mechanism is founded upon a "message bus", where interactions between the monitored system and its environment take the form of messages passed over a common communication mechanism ("bus"). The monitor can be configured to also observe these messages, and so have access to the raw data from which to make its deductions about the activities and state of the monitored system without the need to have access to that system's internals.

The AMOS system implementation operates as described above. We are in the process of further enhancing its usability by adding graphical user interfaces both for entering assumptions to be monitored, and for displaying notifications of assumption violations and other monitored conditions.

ACKNOWLEDGMENTS

This work has been supported initially by the National Science Foundation under award number: DMI-9560648, and subsequently by the Defense Advanced Research Projects Agency under award number: F 30602-96-C-0270. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF, DARPA, or the U.S. Government.

REFERENCES

1. Cohen, D. Compiling complex database transition triggers in *Proc. ACM SIGMOD International Conference on the Management of Data* (Portland, OR, 1989), ACM Press, 225-234.
2. Fickas, S. & Feather, M.S. Requirements Monitoring in Dynamic Environments in *Proc. of the Second IEEE International Symposium on Requirements Engineering* (York, England, U.K., March 1995), IEEE Computer Society Press, 140-147.
3. Girgensohn, Redmiles & Shipman Agent-Based Support for Communication between Developers and Users in Software Design in *Proc of the 9th Knowledge-Based Software Engineering Conference* (Monterey, CA, September 1994), IEEE Computer Society Press, 22-29.
4. Liao, Y. & Cohen, D. PMMS: A Framework and System for High Level Program Monitoring and Measuring in *Proc. of IFIP Congress* (Madrid, Spain, 1992).
5. Mansouri-Samani, M. & Sloman, M. GEM: A Generalized Event Monitoring Language for Distributed Systems. Imperial College of London, Research Report No. DOC 95/8, August 1995.